# Realtime & Streaming Perfect Compression 2

## Description:

Following here is a description of compressing a 15-bit size item. Of course; any item-size may be used by the algorithm; this is just an example description to describe how the algorithm works in general.

We are going to try to write 15 unique bits (2^15) into a shorter bit-string and this without necessary having to make use of a dictionary & DPCM processing and/or variants, but simply by using the sorting-algorithm in bits. As a result, we are not that much dependent on our randomness of our bit-string, and we could compress smaller packages (like win 1 bit out of every 90 bits) and thus get a higher compression rate, suitable for Realtime & Streaming Perfect Compression. This is thus a more advanced version of my original Perfect Compression patent.

### Compression steps:

1) First, we convert our 15 bits, to the 10-base (decimal) cardinal, because we need to know a unique identification of the number representing these bits. With 2^15 possible values, we have 32768 possible combinations (going from 0 to 32767). In other words, our 15-bit number may represent: any number such as 478, or 19054 or 8448,...within the range of 2^15. We may also skip this step and convert our bit-string directly to our desired string (see next step)

2) We use this 10-base (decimal) number, representing the 15-bits, to convert, for instance, to the 8-base cardinal (0-7), and not just this, we also need to make sure that this 8-base cardinal number has unique digits, for instance, the number 10423675 is a possibility.
So, this means that for 8 numbers of 3-bit (8-base cardinal), we have 8! = 8*7*6*5*4*3*2*1 = 40320 unique possible values/combinations that we can represent.
It is of importance that the numbers, the digits representing the number, (in our example: 10423675) are unique, because we want to use a sorting algorithm on them and so there are no numbers missing or doubles (like in 10423474, the 4 is coming up three times and the numbers 5, 6 are missing). In other words, without missing numbers and/or doubles, we don't need any dictionary to know what they represent and thus by using simply our sorting algorithm' bits, we can correctly reform our number (in our example, the number 10423675) Although, not required, we could make use of a dictionary that would enable us know the doubles and/or missing digits in non-unique digits (like in 10423474).

3) Now, we use our sorting algorithm to un-sort 01234567 to 10423675, or 76543210 to 10423675. The resulting bits (of the sorting algorithm) may be fewer than our 15 bits (for instance 14, or 12, then we win, 1 bit and 3 bits respectively)
In other words, the purpose is to find an optimal sorting algorithm to produce the fewest amount of bits in order to un-sort, and thus reproduce our number. We may also use other compression algorithms to further compress this bit-string.
The de-compression part is doing the opposite of these 3 steps, thus recovering our original bit-string.

Also, we need to note that, in this example, our 8 numbers of 3-bits (unique), can represent 8! = 8*7*6*5*4*3*2*1 = 40320 possible combinations, while our 2^15 bits represent only 32768 possible combinations, thus, in effect our 40320, may represent actually 15.3 bits (=log(40320,2)). We thus lose 7552 (= 40320-32768) combinations, or 0.3 bits in the process.
How can we solve this? We could, for instance, combine many of our our 8 numbers of 3-bits

and also in order to come closer to our 2^X number. For example, these 3 combinations: 10423675, 61432570 and 21037654, each represent 15.3 bits, in total = 15.3*3 = 45.9 bits and when combined, they may represent 46 bits, or 2^46 possible combinations). Also, it is not required to convert our original bit-string to an 8-base cardinal number (unique digits), me may, as well, convert it to a 16-base (hexadecimal) cardinal number (unique digits)

De-compression steps:

1) We use our sorting bits, that represent our compressed bit-string, and apply the un-sort algorithm on the number 01234567, in order to get 10423675. Of course, we may also apply the un-sort algorithm to its reverse; like 76543210 or make use of a dictionary that would enable us know the doubles and/or missing digits in non-unique digits.

2) We reverse the process to convert this 8-base cardinal number with unique digits, and convert it to our 10-base (decimal) cardinal number, or convert it directly to our original bit-string (step3)

3) Convert this 10-base (decimal) cardinal number to our binary format, our original 15 bits.

We can further optimize our compression algorithm by using different increment sequences of the sorting algorithm (like in Shellsort). Or simply, by trying different sorting algorithms that results in less comparisons than the one described here. Of course, the compression speed of the chosen algorithm may be as important as the compression rates, since we can continue compressing the resulting data for a second pass and so on...

Now that we can, for instance, compress 90 bits (6 * 15 bits) by 1 bit or more, we can also easily compress big data files (e.g. 1Gb), in blocks of 90 bits. We can now, for instance, use a 'pointer', a number that tells us how many times/steps a file has been compressed by perfect compression. So that each time it decompresses, e.g. from 100 to 99, (we decrement each time by 1), we know the file still has to be decompressed by 99 times. And we may also include a second pointer that tells us in how many blocks the file is compressed; the smaller the compressed file gets, the fewer blocks are used. It is now possible, for example, to compress 1GB of data by an amazing compression factor. Or, compress small 4K-64K clusters for direct use in harddisks or for compressed, streaming data transmission.
Of course, there is a theoretical limit that will tell us how many times a file can be compressed by this perfect compression algorithm, plus we need to store our pointers, telling us how many times a file has been compressed and/or in how many blocks. We may as well decide to compress selectively (like with ShellSort, per 15 bits, choose an n-th part and select the first 7 bits from the bit-string, then the second 8 bits, but 2 bits apart from the first, this continuing until we reaches the end of the file, then we select like 3 bits apart, ...) Or we may also decide to use statistics, for example; if we win 10 bits (with one particular sample), and use only 2^9, or 9 bits to show its position in the file, then we have won 1 bit.

Because the algorithm has a rather small footprint (in memory and length of the algorithm), and we can compress using very small blocks (in this example, blocks of only 90 bits each), the algorithm can easily be programmed on fast parallel FPGA & ASIC circuits (more gates in parallel), for direct implementation in hard disks, modems, satellites, cell phones, etc. As such, we can now achieve Realtime & Streaming Perfect Compression.

time/streaming (random) data, for use to compress video / images, or for use in voice recorders, memory sticks, data banks, external data storage devices, all types of memory cards and external storage media, like Compact Flash, MicroDrive, SmartMedia, SD, MMC, MemoryStick (-Pro), Datapak, Dataflash, Smart Media, USB Memory, PC Card Hard Drives etc.

22. Any application of claims 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 using this technology as software based or hardware based, included but not limited to chips such as FPGA, ASIC, CPLD, EPLD, PLD

23. Any application of claims 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 using this technology in combination with any other data compression algorithms such as those used by GIF, JPEG, ZIP, MPEG, MP3 and others